

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

In re Patent Application of

Atty Dkt. JRL-550-192

NEVILL et al.

C# M#

Serial No. 09/731,060

TC/A.U.: 2194

Filed: December 7, 2000

Examiner: Zhen, Li B.

Date: February 26, 2007

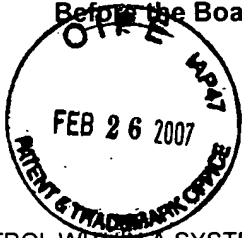
Title: SCHEDULING CONTROL WITHIN A SYSTEM HAVING MIXED HARDWARE AND
SOFTWARE BASED INSTRUCTION EXECUTION

Mail Stop Appeal Brief - Patents

Commissioner for Patents

P.O. Box 1450

Alexandria, VA 22313-1450



Handwritten signature/initials

Sir:

☐ Correspondence Address Indication Form Attached.

☐ **NOTICE OF APPEAL**

Applicant hereby **appeals** to the Board of Patent Appeals and Interferences

from the last decision of the Examiner twice/finally rejecting \$500.00 (1401)/\$250.00 (2401) \$
applicant's claim(s). (\$500 Notice of Appeal fee
previously paid May 23, 2006.)

☒ An appeal **BRIEF** is attached in the pending appeal of the
above-identified application \$500.00 (1402)/\$250.00 (2402) \$ 500.00

☐ Credit for fees paid in prior appeal without decision on merits -\$ ()

☐ A reply brief is attached. (no fee)

☐ Petition is hereby made to extend the current due date so as to cover the filing date of this
paper and attachment(s)
One Month Extension \$120.00 (1251)/\$60.00 (2251)
Two Month Extensions \$450.00 (1252)/\$225.00 (2252)
Three Month Extensions \$1020.00 (1253)/\$510.00 (2253)
Four Month Extensions \$1590.00 (1254)/\$795.00 (2254) \$

☐ "Small entity" statement attached.

Less month extension previously paid on -\$ ()

TOTAL FEE ENCLOSED \$ 500.00

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension.
The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or
asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this
firm) to our **Account No. 14-1140**. A duplicate copy of this sheet is attached.

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:maa

NIXON & VANDERHYE P.C.
By Atty: John R. Lastova, Reg. No. 33,149

Signature: 



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

NEVILL et al.

Atty. Ref.: 550-192

Serial No. 09/731,060

Group: 2194

Filed: December 7, 2000

Examiner: Zhen, Li B.

For: SCHEDULING CONTROL WITHIN A SYSTEM HAVING MIXED
HARDWARE AND SOFTWARE BASED INSTRUCTION
EXECUTION

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT

**On Appeal From Final Rejection
From Group Art Unit 2194**

John R. Lastova
NIXON & VANDERHYE P.C.
11th Floor, 901 North Glebe Road
Arlington, Virginia 22203-1808
(703) 816-4025
Attorney for Appellants
Nevill et al and
ARM Limited



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

NEVILL et al.

Atty. Ref.: 550-192

Serial No. 09/731,060

Group: 2194

Filed: December 7, 2000

Examiner: Zhen, Li B.

For: SCHEDULING CONTROL WITHIN A SYSTEM HAVING MIXED
HARDWARE AND SOFTWARE BASED INSTRUCTION
EXECUTION

February 26, 2007

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a United Kingdom corporation.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals related to this subject application. There are no interferences related to this subject application.

02/27/2007 MBERHE 00000109 09731060

01 FC:1402

500.00 0P



III. STATUS OF CLAIMS

Claims 1-16 are pending and rejected.

IV. STATUS OF AMENDMENTS

No amendment was filed after final. A pre-appeal was filed.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Multitasking operating systems require processing resources to be shared between several different programs that may be simultaneously active, and multithreaded computer programs similarly require processing resources to be shared between different active threads. To support execution of higher level computer program languages, mixed hardware-based execution units and software-based execution units may be used. Simple instructions may be executed under control of that hardware-based execution unit, whereas more complex program instructions trigger the execution of a software routine, typically written in a lower level, directly executable program language, which interprets the complex instructions. Although such systems can provide comprehensive and relatively high speed execution of high level program instructions, they pose difficulties in also supporting reliable scheduling between tasks or threads.

The claims describe an apparatus and a method to reliably and efficiently support scheduling within mixed hardware and software based instruction

execution systems. Program instructions are sent to the hardware based instruction execution unit and forwarded from there to the software based instruction execution unit if they cannot be dealt with by the hardware based instruction execution unit. By routing all the program instructions through the hardware based instruction execution unit, it can keep track of the execution of instructions and generate a scheduling signal for triggering a scheduling operation irrespective of whether the preceding instructions have been executed by hardware or software. This reduces the likelihood of scheduling operations being triggered at inappropriate times and thus avoids computational errors that could potentially arise due to inappropriately triggered scheduling. It also provides an efficient way of managing scheduling relative to known counter-based scheduling techniques since it avoids overheads involved with maintaining and exchanging a plurality of counters.

Claim 1 recites an apparatus for processing data operable to execute operations specified in a stream of program instructions.¹ Figure 5 shows a data processing system 102 including a processor core 104 and a register bank 106. An instruction translator 108 is provided within the instruction path to translate Java Virtual Machine instructions to native instructions (or control signals corresponding thereto) that may then be supplied to the processor core 104. The examples describe the native instructions as ARM instructions. The instruction

translator 108 may be bypassed when native ARM instructions are being fetched from the addressable memory. See page 10, lines 6-11. Figure 9 shows a Java bytecode translation unit 64 that receives a stream of Java bytecodes (an example of program instructions) and outputs a translated stream of native ARM instructions (or corresponding control signals) to control the action of a processor core. Page 26, lines 30-32.

Simple Java bytecodes are executed by high speed hardware based translation. See the left side of Figure 10 under “HARDWARE”, steps 80 and 82, and page 27, lines 5-6. Program instructions for which hardware based execution is not supported, e.g., bytecodes requiring more complex processing operations, are sent to a software interpreter. See the right side of Figure 10 under “SOFTWARE”, block 84, and page 27, lines 6-9. More specifically, the Java bytecode translation unit 64 determines that a received bytecode is not supported by hardware translation, and a branch is made to an address where a software interpreter routine is found or referenced. After the complex bytecode has been executed using the software interpreter (block 84), control is returned to the hardware based execution (Figure 10 and page 27, lines 31-34).

Scheduling logic is provided in the hardware based execution unit. A non-limiting example is the counter 70 in the bytecode translator 64 shown in Figure 9. All Java bytecodes are received by the Java bytecode translation unit 64 and cause

¹ The following explanation is in the context of non-limiting examples. The claims are not limited to these

the counter 70 to be decremented. See step 72 in Figure 10. If the counter value has reached 0, irrespective of whether a preceding program instruction was executed by the hardware based execution unit or the software based execution unit, then the hardware based execution unit generates a scheduling signal for triggering a scheduling operation to be performed between bytecodes, e.g., a branch is made from the hardware based translator 64 to scheduling code as shown by the dashed line in Figure 9 and the arrow leading out from block 74 to step 76 (“Do Processing Scheduling”) in Figure 10. Page 27, lines 19-22. Once the scheduling code has completed the scheduling at step 76, control is returned to the hardware and processing proceeds to back to step 72, where the next Java bytecode is fetched and the counter again decremented. Page 27, lines 22-24. Figures 12 and 13 describe an alternative example embodiment. Page 28, line 16-page 29, line 5.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The rejection on appeal is that of claims 1-16 under 35 U.S.C. §103 as being obvious based upon Evoy (5,937,193) and Gee (6,374,286).

VII. ARGUMENT

A. The Legal Requirements For Obviousness

An invention is obvious only "if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains." 35 U.S.C. §103. Obviousness is a legal conclusion based on underlying findings of fact. *In re Dembiczak*, 175 F.3d 994, 998 (Fed. Cir. 1999). The underlying factual inquiries are: "(1) the scope and content of the prior art; (2) the level of ordinary skill in the prior art; (3) the differences between the claimed invention and the prior art; and (4) objective evidence of nonobviousness." *Id.* Determination of obviousness cannot be based on the hindsight combination of components selectively culled from the prior art to fit the parameters of the patented invention. *ATD Corp. v. Lydall, Inc.*, 159 F.3d 534, 546 (Fed. Cir. 1998). There must be some teaching, suggestion, or reason in the prior art to select particular elements, and to combine them as combined by the inventor. *Vulcan Engineering Co., Inc. v. Fata Aluminium, Inc.*, 278 F.3d 1366, 1372 (Fed. Cir. 2002). "The absence of such a suggestion to combine is dispositive in an obviousness determination." *Gambro Lundia AB v. Baxter Healthcare Corp.*, 110 F.3d 1573, 1579 (Fed. Cir. 1997).

B. The Combination of Envoy and Gee Fail To Teach All Claim Features

1. The Evoy and Gee Patents

Evoy describes a translating circuit that translates platform-independent instructions such as Java bytecodes into corresponding native instructions for execution by the processor. A state machine uses look-up tables to map platform-independent instructions into native instructions for the processor to minimize software-based interpretation of platform-independent program code on-the-fly or in blocks prior to execution. An exception signal is provided as a data signal or an interrupt signal to allow the processor to return to native mode and interpret an unmapped bytecode via a software interpreter. Evoy at 5; 64-67 and 6; 14-16.

Gee describes multiple Java Virtual Machines (JVMs) operating on a single direct execution JAVA processor with each JVM operating in a separate time slice called a partition. Each JVM has its own data and control structures and is assigned a fixed area of memory. Each partition is also allotted a fixed period of time in which to operate, and at the end of the allotted time, a context switch is forced to another JVM operating in the next partition. The context switch does not transfer control directly from one JVM to another JVM. Instead, at the end of a partition time period control is switched from the currently operating JVM to a "master JVM" during a time period called an "interslice." The master JVM handles system interrupts and housekeeping duties. At the end of the interstice

time period, the master JVM starts a proxy thread associated with the next JVM to become operational. The proxy thread handles JVM-specific interrupts and checks the status of the associated JVM. If the JVM appears operational the proxy thread transfers control to the JVM thread.

2. Neither Evoy Nor Gee Teach Element (iv) In The Independent Claims

Neither Evoy nor Gee discloses the independent claim feature (iv) where program instructions that are not supported by the hardware based execution unit are “forwarded to said software based execution unit for execution with control being returned to said hardware based execution unit for a next program instruction to be executed.” The Examiner relies solely on Evoy and contends that “selecting the next bytecode” in col. 7, lines 8-16 of Evoy (7:8-16) discloses the quoted claim feature. Appellant disagrees.

Evoy teaches at 6:61-7:21 that although the processor normally runs in native mode, when the processor requests instructions from the platform-independent partition of the system memory, the processor 40 switches to a platform-independent mode. An object table 51 is scanned to see if a stored native instruction corresponds to a given bytecode. If so, after executing the corresponding native instruction in the platform-independent mode, the processor increments its address counter to the next instruction which has the effect of selecting the next bytecode (7:17-20). But if no corresponding native instruction

exists, the object table 51 outputs an exception signal which notifies the processor that software interpretation of the bytecode may be required (7:14-16). When software interpretation occurs, the processor returns to native mode to execute native software instructions in native mode. See Evoy at 5:64-67 and 7:27-29. After executing the complex bytecode in the native mode, Evoy does not disclose when and if control is returned to the platform independent mode.

Evoy's "selecting the next bytecode" text, upon which the Examiner relies, only applies when the processor is in the platform-independent mode—not when the processor is in native mode. Otherwise, why would the next bytecode—a non-native instruction—be selected? When the processor encounters a non-native bytecode instruction it cannot handle when in the platform-independent mode, software interpretation of the bytecode is performed in the native mode. In that native mode, the processor accesses native instructions to implement the complex non-native bytecode instruction. See 5:64-67 and 7:27-29. When the processor is in its normal native mode, the statements regarding "selecting the next bytecode" (7:20) do not apply. In native mode, the next native instruction is selected and not the next non-native bytecode.

Thus, contrary to the Examiner's assertion, 7:8-16 of Evoy does not disclose the claimed feature where control is returned to the hardware based execution unit for a next program instruction to be executed. When Evoy's processor is in native mode, the address counter points to native instructions and

incrementing the address counter selects the next native instruction for execution.

Control is not returned to the non-native, platform independent mode once software interpretation has been invoked.

The Examiner contends that the translation circuit 50 of Evoy is a counterpart of the claimed hardware based instruction execution unit. Evoy teaches that the non-native, platform-independent mode (in which translation circuit 50 and object table 51 are used) is only entered when the processor requests instructions from a platform-independent partition of system memory (7: 1-7). As just explained, return from non-native, platform independent mode to native mode occurs when a native instruction does not exist that can be directly mapped to the addressing bytecode in the object table 51. Specifically, an exception is generated to allow the processor to return to native mode and interpret the unmapped bytecode via a software interpreter. But Evoy never teaches returning control to the hardware-based execution unit, corresponding to the translation circuit 50 used in the platform-independent mode of Evoy, after a bytecode has been forwarded to the software based execution unit (the software interpreter) for execution.

The Examiner also refers to 11:6-15 of Evoy which discloses that if the “source address” value does not equal the “source end address” value, then additional platform-independent instructions must be processed. Once the source address register has been processed, control returns to block 202 to process the next platform-independent instruction. This passage can be understood with

reference to Figure 6. The text at 10:32-34 explains that Figure 6 relates to a translation code routine for a translation state machine. So this 11:6-15 passage relates translating and storing Java bytecodes for subsequent execution. No Java bytecodes are actually executed in the flow chart of Figure 6. Accordingly, 11:6-15 cannot disclose the claimed feature where "control is returned to the hardware based execution unit for the next program instruction to be executed."

3. Combining Gee With Evoy Gee Does Not Result In Claim Element (v)

The broadest reasonable interpretation standard applied by the Examiner is not reasonable because it ignores language in the claim. On page 2 of the September 21, 2006 office action, the Examiner interprets element (v) of claim 1 as follows:

the hardware based execution unit includes scheduling support logic operable to generate a scheduling signal for triggering a scheduling operation to be performed by scheduling code. The scheduling code (software), manages between program instructions for managing scheduling between threads or tasks irrespective of whether a preceding program instruction was executed by the hardware based execution unit or the software based execution unit.

But the Examiner has rewritten this claim element by adding the underlined words above and leaving out words underlined below from the actual claim language:

said hardware based execution unit includes
scheduling support logic operable to generate a

scheduling signal for triggering a scheduling operation
to be performed between program instructions for
managing scheduling between threads or tasks
irrespective of whether a preceding program
instruction was executed by said hardware based
execution unit or said software based execution unit.

As clearly stated in claim feature (v), the hardware based execution unit includes scheduling support logic that generates “a scheduling signal for triggering a scheduling operation to be performed....” It is improper for the Examiner to add to or remove words from the claims.

In any event, the Examiner concedes that Evoy only discloses scheduling in a general sense in that an address counter is incremented after the execution of an instruction, which has the effect of selecting the next bytecode. As the Examiner admits, incrementing an address counter to select the next bytecode for translation is not scheduling between threads or tasks.

Multi-tasking operating systems require processing resources to be shared between several different programs that may be simultaneously active, and multi-threaded computer programs similarly require processing resources to be shared between different active threads. The inventors recognized drawbacks with simplistic counter or timer scheduling approaches. A simple timer-based scheduling approach may suffer from the disadvantage that scheduling operations may be inappropriately triggered at points part-way through the software interpretation of a complex program instruction in a manner that could cause a loss

of data integrity should an inappropriate context switch occur. A simple counter-based scheduling requires the extra overhead of exchanging counter values between hardware-executed program instructions and software-executed program instructions. Indeed, Evoy suffers from these very problems where scheduling operations may be triggered part way through interpretation of a complex non-native program instruction.

The Examiner relies on text in Gee at 21:25-60 which simply discloses a "**conventional** java scheduling policy" (21:30-31 (emphasis added) where threads are scheduled to run in accordance with priority-based rules. All Gee describes is that the highest priority thread that is ready to be executed is dispatched from the ready thread queue. Gee's conventional scheduling policy is just another example of a known scheduling policy. Gee does not disclose a scheduling policy like that claimed.

The independent claims recite: (1) returning control to the hardware based execution unit for a next program instruction to be executed (as explained above) and (2) the hardware based execution unit triggering a scheduling operation between program instructions for managing scheduling between threads or tasks irrespective of whether a preceding program instruction was executed by the hardware based execution unit or the software based execution unit. The issue of managing scheduling "irrespective of whether a preceding program instruction was executed by the hardware based execution unit or the software based

execution unit” does not arise in Gee because Gee does not disclose both the hardware-based and software-based execution units. So the Examiner lacks the proper claimed context to argue that Gee performs scheduling “irrespective of whether a preceding program instruction was executed by said hardware based execution unit or said software based execution unit.”

The “irrespective ...” clause in claim element (v) must be consider together with the “control being returned to said hardware based execution unit” clause of claim element (iv). In the independent claims, it is because control is returned to the hardware execution unit after a program instruction has been executed by the software-based execution unit that the scheduling operation can be triggered “irrespective of whether....” The Examiner wrongly analyzes element (v) recited in the independent claims in isolation rather than in the whole context of the claim as required.

A benefit of this claimed approach not found in either Evoy, Gee, or their combination is that the hardware-based execution unit can keep track of the execution of instructions, and as a result, reliably generate the scheduling signal for triggering a scheduling operation irrespective of whether the preceding instructions have been executed by hardware or software. Even if one included Gee’s conventional highest priority thread scheduling in Evoy, the result would still suffer from the possibility that the highest priority thread may be scheduled

part way through software interpretation of a complex non-native program instruction.

4. Evoy and Gee Do Not Address or Solve the Problems Solved in This Case

Evoy and Gee do not recognize the problems to which the missing features are directed where “conventional” scheduling operations may be inappropriately triggered (simple timer-based approach) or have disadvantageous overhead of counter exchange (counter-based scheduling) in processing systems having mixed hardware based execution units and software based execution units. Nor would implementing Gee’s the scheduling policy in Evoy’s system solve these problems or teach the combination of features specified by independent claims. The failure to recognize the problem being solved by the claims is evidence that the required “teaching, suggestion, or motivation to combine the references” is missing. *In re Rouffet*, 149 F.3d 1350, 1355 (Fed. Cir. 1998). Specifically, the Examiner’s proposed combination does not solve the problems with permitting scheduling regardless of whether the preceding program instruction was executed by the hardware based execution unit or the software based execution unit.

In contrast, the combination of claim features (iv) and (v) prevents this problem and maintains synchronization between hardware and software instruction execution with respect to a scheduling signal. Control is returned to the hardware based execution unit even if the program instruction to be executed

has been forwarded to the software based execution unit for execution. Routing the program instructions for execution to the hardware based execution unit and returning control to the hardware based execution unit enables the hardware unit to trigger the scheduling operation irrespective of whether a preceding program instruction was executed by the hardware based execution unit or the software based execution unit.

What follows is an example that demonstrates the significance of the claim features (iv) and (v) where the scheduling operation is "performed between program instructions."

Consider a platform independent operation (*bytecode*)INC. The operation INC increments a variable by performing a native LOAD, ADD, and STORE operation. Thus, INC <variable> would be converted to

LOAD <register>, <variable>

ADD <register>, #1

STORE <register>, <variable>

Now consider the case where <variable> initially has the value 1 and an interrupt (or scheduling operation) occurs after the first LOAD. A second thread is then scheduled which performs an INC operation on the same <variable>. Some time later the first thread is rescheduled and completes the first INC operations.

Thread 1 LOAD <register[1]>, <variable> ; <register[1]> = 1

```
; <- interrupt, Thread 2 is scheduled
```

```
Thread 2      LOAD <register[2]>, <variable> ; <register[2]> = 1
              ADD  <register[2]>, #1          ; <register[2]> = 2
              STORE <register[2]>, <variable> ; <variable> = 2
              .....                          ; More instructions from Thread 2
              ; <- interrupt, Thread 1 is scheduled
```

```
Thread 1    ADD  <register[1]>, #1          ; <register[1]> = 2
            STORE <register[1]>, <variable> ; <variable> = 2
```

Note: the denotation $\langle \text{register}[1] \rangle$ and $\langle \text{register}[2] \rangle$ means the $\langle \text{register} \rangle$ belonging to the context of thread 1 and 2 respectively.

At the end of this execution sequence in a system such as that proposed by the Examiner based on Evoy and Gee, the <variable> could have the value 2 instead of 3. Now consider what happens with the benefit of the system defined in claim 1:

```
Thread 1      LOAD      <register[1]>, <variable>    ; <register[1]> = 1
                                     ; <- interrupt
                                     ; However, scheduling operation is
                                     ; deferred because INSTRUCTION
                                     ; is de-asserted (see Fig 11 and text)

      ADD <register[1]>, #1          ; <register[1]> = 2

      STORE <register[1]>, <variable> ; <variable> = 2
```

; INSTRUCTION is now asserted

; and the scheduling operation

; permitted.

Thread 2 LOAD <register[2]>, <variable> ; <register[2]> = 2

```
ADD <register[2]>, #1      ; <register[2]> = 3
```

```
STORE <register[2]>, <variable> ; <variable> = 3
```

```
....; More instructions from Thread 2
```

```
; <- interrupt, Thread 1 is scheduled
```

Thread 1 ... ; More instructions from Thread 1

At the end of this execution sequence, <variable> correctly has the value 3.

According to claims 1 and 16, the program instructions are sent to the hardware based execution unit and forwarded to the software based execution unit, with control being returned to the hardware based execution unit, to ensure that a scheduling operation will not be performed in the middle of execution of the Java bytecode, which often requires the execution of several native instructions.

5. Dependent Claim Features Are Not Taught

The Examiner contends that Evoy 7:17-27 discloses the subject-matter of claim 2. But address counter (or program counter) which is completely different from the claimed counter included in the scheduling support logic. Evoy's address counter is a pointer to a memory location holding the current instruction, which is

incremented after execution of each instruction. This address counter does not count up or down towards any specific value. It would not be possible to use Evoy's address counter as scheduling support logic because it could not be used to program a scheduling signal to occur when the counter reaches a particular value.

The Examiner contends that the "END reserved code" of 10:65-11:6 of Evoy corresponds to the subject-matter of claim 3 where the counter triggers generation of the scheduling signal when the predetermined count value is reached. The Examiner equates the END reserved code of Evoy with the predetermined count value. But this passage does not teach generating a scheduling signal "for managing scheduling between threads or tasks" as recited in claim 1. Instead, this passage discloses passing control to process a new instruction in the event that the new instruction is not an END reserve code. This is different from generating a scheduling signal when a counter has reached a predetermined count value as specified by claim 3.

VIII. CONCLUSION

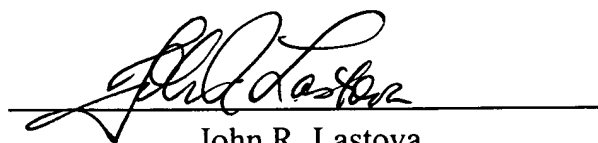
Claim features missing from the combination of Evoy and Gee. The obviousness rejection is also legally improper because neither reference acknowledges or solves the problem to which the claims in this application are directed. The prior art rejection should be reversed, and this application passed to allowance.

Nevill et al Appeal
Serial No. 09/731,060

Respectfully submitted,

NIXON & VANDERHYE P.C.

By:

A handwritten signature in black ink, appearing to read "John R. Lastova", is written over a horizontal line.

John R. Lastova
Reg. No. 33,149

JRL/maa
Appendix A - Claims on Appeal

IX. CLAIMS APPENDIX

1. (Previously Presented) Apparatus for processing data operable to execute operations specified in a stream of program instructions, said apparatus comprising:
 - (i) a hardware based instruction execution unit operable to execute program instructions; and
 - (ii) a software based instruction execution unit operable to execute program instructions; wherein
 - (iii) program instructions to be executed are sent to said hardware based execution unit for execution;
 - (iv) program instructions received by said hardware based execution unit for which execution is not supported by said hardware based execution unit are forwarded to said software based execution unit for execution with control being returned to said hardware based execution unit for a next program instruction to be executed; and
 - (v) said hardware based execution unit includes scheduling support logic operable to generate a scheduling signal for triggering a scheduling operation to be performed between program instructions for managing scheduling between threads or tasks irrespective of whether a preceding program instruction was executed by said hardware based execution unit or said software based execution unit.

2. (Original) Apparatus as claimed in claim 1, wherein said scheduling support logic includes a counter with a value that is changed in response to a program instruction sent to said hardware based execution unit.
3. (Original) Apparatus as claimed in claim 2, wherein said counter triggers generation of said scheduling signal when a predetermined count value is reached.
4. (Original) Apparatus as claimed in claim 3, wherein said counter may be programmed to start from a user programmable start value.
5. (Original) Apparatus as claimed in claim 3, wherein said counter counts up to said predetermined value.
6. (Original) Apparatus as claimed in claim 3, wherein said counter counts down to said predetermined value.
7. (Original) Apparatus as claimed in claim 1, wherein a debug operation is triggered by said scheduling signal.
8. (Original) Apparatus as claimed in claim 1, further comprising timer logic operable to generate a timer signal indicative of a time since a last scheduling operation.

9. (Original) Apparatus as claimed in claim 8, wherein said scheduling signal is combined with said timer signal to trigger said scheduling operation.

10. (Original) Apparatus as claimed in claim 8, wherein a scheduling operation is triggered upon generation of said scheduling signal after said timer signal has reached a predetermined value indicating a predetermined period time since a last scheduling operation has expired.

11. (Original) Apparatus as claimed in claim 1, further comprising a processor core operable to execute operations as specified by instructions of a first instruction set.

12. (Original) Apparatus as claimed in claim 11, where said hardware based instruction execution unit includes an instruction translator operable to translate instructions of a second instruction set into translator output signals corresponding to instructions of said first instruction set.

13. (Original) Apparatus as claimed in claim 12, wherein

(i) at least one instruction of said second instruction set specifies a multi-step operation that requires a plurality of operations that may be specified by instructions of said first instruction set in order to be performed by said processor core; and

(ii) said instruction translator is operable to generate a sequence of translator output signals to control said processor core to perform said multi-step operation.

14. (Original) Apparatus as claimed in claim 1, wherein said software based execution unit is a software based interpreter.

15. (Original) Apparatus as claimed in claim 1, wherein said program instructions are Java Virtual Machine instructions.

16. (Previously Presented) A method of processing data by executing operations specified in a stream of program instructions, said method comprising the steps of:

(i) executing program instructions with a hardware based instruction execution unit; and

(ii) executing program instructions with a software based instruction execution unit; wherein

(iii) program instructions to be executed are sent to said hardware based execution unit for execution;

(iv) program instructions received by said hardware based execution unit for which execution is not supported by said hardware based execution unit are forwarded to said software based execution unit for execution with control being returned to said hardware based execution unit for a next program instruction to be executed; and

(v) said hardware based execution unit generates a scheduling signal for triggering a scheduling operation to be performed between program instructions for managing scheduling between threads or tasks irrespective of whether a preceding program instruction was executed by said hardware based execution unit or said software based execution unit.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.